EVALUATION OF SURROGATE MODELING AS A WAY TO REDUCE THE COMPUTATIONAL BURDEN OF NUMERICAL SIMULATIONS [1]

Mark A. Cowan* and Dr. Cary D. Butler
U.S. Army Engineer Research and Development Center
Information Technology Laboratory
3909 Halls Ferry Road
Vicksburg, Mississippi 39180-6199

INTRODUCTION

Real-world application domains such as engineering design problems are computationally expensive. Some problems require thousands of simulations, which places a major burden on computing assets. Today's machine capacities have helped by affording users the ability to run numerous scenarios to discover a range of possible outcomes and to understand the influence of parameter variation on a problem. These activities become problematic as scenario sizes and complexities continuously increase. As a result, the high-performance computers often suffer at the hands of a combinatoric explosion.

A single simulation may require hours to days to complete, and an exhaustive ("brute force") exploration of the design space for optimization purposes and "what-if" analyses can easily call for thousands of simulations. An extreme example involves numerical simulations for a hurricane study for the Gulf of Mexico that was assigned ~4 million CPU hours per year over 3 years. This would require full utilization of over 500 CPUs. This results in millions of dollars being spent just for computing time alone. The effort proceeded in brute-force manner, exhaustively solving over an immense problem space, often with very minor tweaks to the initial conditions.

This paper proposes a new approach to analyzing engineering design problems, replacing the numerical simulations with an emulation capability termed a surrogate model. Developing a surrogate model requires an accurate definition of the mapping between all of the interesting input values (i.e., the input space) and the associated output of the numerical simulation (i.e., the output space). The approach assumes that the number of simulations required to sufficiently cover this mapping is much smaller than assessing all possible conditions. A significant challenge in this approach involves the process whereby the salient factors driving the real-world phenomena are teased out through sensitivity analysis and expert insight. A divide-and-conquer strategy (input bisection) is used to subdivide the input space (within an acceptable error bound), which leads to significant reductions in the number of simulations required. The partitioning of the input space is driven by the sensitivity of the output space to changes to the initial conditions of the simulation. Consequently, finer levels of partitioning occur in sections of the input space that produce significant changes to the output space. Conversely, sections of input space that result in minimum changes to the output are assessed at a much coarser level.

When the partitioning of the input space to locate the minimum number of input-output pairs required to properly represent the mapping of input to output is completed, an artificial neural network (ANN) is trained. The ANN has the ability to interpolate the numerical model's reaction to scenarios that were originally excluded through the partitioning algorithm. A properly trained ANN provides inferencing capabilities for the surrogate model.

Visually interacting with the surrogate model provides the ability to explore the problem space and quickly pinpoint the critical model parameters without the expense of running hundreds of unnecessary simulations. The surrogate model is refined by running additional input/output mapping cases, lowering error thresholds, and repeating the ANN training. This process is repeated until an acceptable approximation of the problem is obtained.

1. OVERVIEW OF SURROGATE MODEL APPROACH

The purpose of this project is to devise methodologies/techniques that are elegant, widely applicable across many different solvers, and easy to implement. These methodologies will significantly reduce the number of high performance computer (HPC) runs necessary to acquire solution approximations within an acceptable level of accuracy. This paper presents a five-step approach to constructing a surrogate model:

- Configure a numerical model M_D for a problem domain D.
- Define the set of all possible parameters P that define the initial condition of the model. A subset of the parameters, identified as p, is classified by the domain expert as critical parameters for the simulation.

[1] Approved for public release; distribution is unlimited.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
DEC 2008	**N/A**	**-**

4. TITLE AND SUBTITLE
Evaluation Of Surrogate Modeling As A Way To Reduce The Computational Burden Of Numerical Simulations

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center Information Technology Laboratory 3909 Halls Ferry Road Vicksburg, Mississippi 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002187. Proceedings of the Army Science Conference (26th) Held in Orlando, Florida on 1-4 December 2008, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**8**	

- Represent the geometry G that defines the spatial makeup of D.
- Represent a series of time-steps T predicted by M_D.
- Define variations of the input space (defined as I) as defined by the product set of p, G, and T.
- Define the output space O by all possible results of M_D over I.

Therefore the goal of this effort is to develop an emulator of M_D, termed surrogate model SM_D, that has the ability to map I to O with similar accuracies to those of M_D. Acceptability of SM_D is based on the error tolerances provided by the domain expert. Error is defined as the differences between M_D and SM_D in mapping I to O.

The underlying algorithm for the SM_D is an artificial neural network (ANN). Several variations of ANN algorithms have been evaluated; however, for this paper the work is based on the backpropagation neural network algorithm. Various configurations of the hidden layer have been evaluated that include changes from the number of nodes to multiple hidden layers.

The five-step process for creating an SM_D is outlined as follows:

- **Step 1 – Data Preparation:** The goal of this step is to identify a subset of I, denoted i, that best represents the input space in I. Ideally $|i|$ is significantly smaller than $|I|$. The reduction process by which i is created involves filtering I based on data reduction algorithms applied to p, G, and T.
  - *p reduction* – Reduction based on p follows from a bisection algorithm that continuously subdivides p until the variations in M_D output are within tolerance. Partitioning is more prone to occur in the complex sections of the input space. For each parameter in p, domain experts provide a value that indicates the precision used to subdivide each dimension in the parameter space.
  - *G reduction* – The next step involves reducing the geometry G by downsampling the number of nodes that will be considered during training. The resulting geometry is represented by g. Note that the reduction occurs only on the interior nodes. The boundary nodes are left intact.
  - *T reduction* – The final step involves downsampling the time steps to create a subset t. Reducing the number of time-steps considered occurs by having a domain expert provide a distribution of importance versus time. Importance is defined on an interval [0,1]. Each time-step is considered for inclusion into i based on its probability relative to 1—i.e., its importance rating.
- **Step 2 - Configuring the ANN:** One of the challenges involves properly configuring the ANN. A small number of neurons has limited memory and underfits the data, while one with too many neurons has sufficient memory and possibly overfits or memorizes the data. The objective is to identify an ANN configuration that best approximates over I, and yet generalizes well on unseen examples selected across I.
- **Step 3 - Training the ANN:** This step involves selecting ~90 percent of the cases (training data) identified in i for training purposes. Training continues until the approximation ability of the ANN reaches a predefined error threshold e_t or convergence, whichever comes first.
- **Step 4 - Validating the SM_D against M_D:** Validation occurs by running the ANN over a set of cases in i that were not used during training (testing data). This provides an indication of the network's ability to predict outside cases used for training. Secondly, the ANN can be compared to M_D based on random cases selected across I. Finally, the M_D and SM_D can be connected to visualization tools that allow domain experts to see a side-by-side comparison of both models running on selected cases of I. Results that are unacceptable require additional work in Steps 1 or 2.
- **Step 5 - Constructing the SM_D:** After the validation of the ANN is completed, the next step involves packaging it into a usable form. During this step, the goal is to develop the ability to interface with existing input files used by M_D and tailor the output as required by the end users. SM_D s are implemented as Windows applications with user controls for varying the input values represented as p and time represented as t. A graphical rendering of the SM_Ds output reacts as the user varies p and t.

2. FROM NUMERICAL MODEL TO SURROGATE MODEL COUNTERPART

Generally, the input space I to numerical solvers defines the physics parameters P (usually through boundary or initial conditions), the spatial aspects G (through a geometry file associating various Cartesian points in space, say [x,y,z], and their connectivities), and the temporal dimension T (by enumerating the various time-steps for which a solution is desired). Once defined, a single set of physics parameters, geometry, and time configuration is sent into the numerical solver for processing. Upon successful completion, the numerical model will issue an output value (or a set of values) for each point in the geometry file for each time-step represented as O. For example, within a temperature problem, a single temperature will be returned for each point within the geometry for each time-step, say, in

10 minute increments throughout a full day (midnight to midnight). Of course, the wide range of variation in defining *I* can (and usually does) result in an enormous number of possibilities, and limitations—whether due to time, costs, or resources—may preclude investigating all of these combinations. A strategy must be delineated t0 permit the solver to focus its efforts upon that portion of the input space (whether it be the parameters, geometry, time, or some significant combination thereof) and capture the solutions there, while expending considerably less effort on less interesting (or less challenging) portions of the input space.

Consider the requirements of an ANN trying to learn simple input-output pairs. Here the backpropagation algorithm is described. Other architectures of neural networks and other machine learning techniques may work as well as or even better than the backpropagation algorithm, depending upon the context of the underlying physics problem. A collection of input-output pairs is divided into two subsets: a training set (usually comprising 90 percent of the collection) and a testing set (comprising the remaining 10 percent). The neural network runs through, say, 10,000 epochs where it receives the training input, adjusts its weights internally, and tries to predict the output. The known output is compared with the predicted output, and the neural network's weights are then adjusted to minimize error. Over time the network can often be trained to emulate the training portion of the input-output set within a user-defined error bound. Of course, noisy input sets can make this much more difficult than more well-behaved sets and may therefore require more training epochs to converge within the error bounds.

Upon the completion of training, the weights (now assumed as representative of the training set) are used to test the neural network against the testing set. Generally, since this set has never been seen and hence the neural network has no history of training against this data, the error rate should be somewhat higher than the corresponding error rate on the training set. Although in theory these input sets are as generic as the input parameters to any function, they can be used to reflect the geometry, the boundary (or initial) conditions, time, and the associated output of a numerical solver. That is, the ANN can be trained to serve as a surrogate model for the numerical solver. The ANN in this case would adjust its internal weights to imitate the physics being modeled within the numerical solver. The input and output sets play the same roles in both tools. Fig. 1 shows some commonalities in the relationship between numeric solvers and artificial neural networks.

As shown above, the backpropagation algorithm when trained properly over input-output sets from a numerical solver can serve as a function approximator to

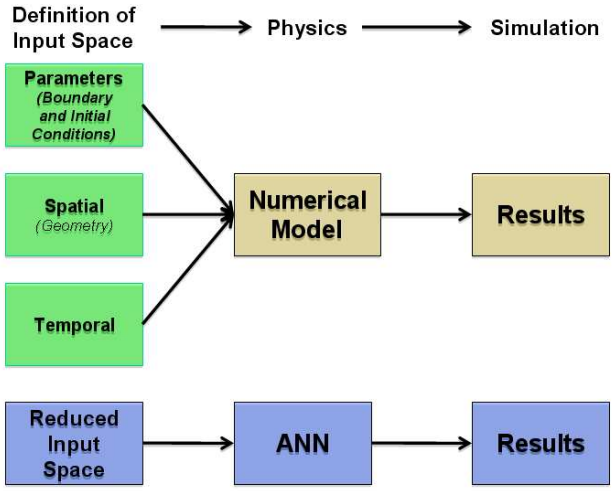

Fig. 1.

the solver within the ranges of training. For instance, backpropagation neural networks have served as interpolation tools for decades within control theory applications. The techniques work no differently here.

However, for all of the obvious parallels between the numeric solver and the neural network, the differences should not be ignored. The neural network as described does not work apart, in a vacuum, away from the results of the solver. Instead it uses the input and output of the solver to imitate the solver. In other words, it does not replace the numerical solver; it complements it and reduces its use to a bare minimum. Or better yet, it provides a second path to numerical solver quality answers without the computational burden of actually running the numerical solver. These input-output pairs being used to train the neural network should collectively be representative of the entire input parameter space across all of the categories: boundary (or initial) conditions, geometry, and time. A loss of representativeness in any of these areas handicaps the ANN and reduces its effectiveness along that dimension of concern. These surrogate models cannot imitate solvers for which the input-output pairs are missing. This word-of-warning cuts both ways—in demanding representativeness of the whole parameter space, and in refusing any attempts at extrapolation outside the range of training.

3. DATA REDUCTION OF NUMERICAL SOLVER OUTPUT

A healthy juxtaposition of divergent aims exists here. On the one hand, running the numerical solver exhaustively over the full input space to the finest resolution of parameter inputs would certainly be representative of the input space. It would be costly and a poor exercise in resourcefulness, but assuredly representative. On the other hand, insight is required into

the input space. A handful of numerical solver runs is probably insufficient to provide the information for an approximator to the numerical solver within the desired error bounds. So this case would be highly efficient, but information-poor. The goal is to identify a minimum set of parameter value combinations that sufficiently represent the input space. Within the input space, regions may exist in which it is relatively easy to predict the dynamics of the problem (and hence the output of the numerical solver); this area should be sparsely sampled. Other regions in the input space may require seemingly inordinate amounts of numerical solver runs to adequately reflect the dynamics involved; this area accordingly should be highly sampled. The distinction between these types of regions is essential to any reduction in the number of numerical solver runs.

A bisection routine was used to exploit this desire for high-resolution insight into these areas of very involved dynamics while sampling sparsely in the regions of simpler dynamics. Other techniques such as gradient methods could have been substituted for the bisection method and may prove somewhat more efficient in particular cases; but bisection seems reasonable, relatively efficient, widely applicable across very different types of numerical solvers, easy to implement, and requiring little formal justification to users.

The technique in one dimension proceeds as follows: Consider parameter p_i, from the set of input parameters P, regarded as a driving factor in the numerical solver. Parameter p_i has minimum m and maximum M. The user desires outputs from the numerical solver, say, no further than 0.3 units apart. The numerical solver is run twice, once for parameter p_i equal to its minimum m and again for parameter p_i equal to its maximum M. Now the user has the output values for each point in space for each time-step, under these divergent input conditions m and M. The output sets are compared point by point to determine if the maximum difference over all points over all times is greater than the 0.3 units desired. (Here an average, a standard deviation, or comparison over just a handful of key points is possible, depending on the context of the problem.) Suppose at least one such maximum exists that exceeds the 0.3 units. Then the midpoint $(m + M)/2$ is used as the value of parameter p_i in the execution of the numerical solver. Similarly, its output is compared with those of the preceding runs to its left and right, and a decision is made whether to subdivide again. In so doing, the bisection method builds a tree that reflects the necessities of numerical solver runs over the parameter space for this single variable p_i. The same technique carries over into multiple dimensions, each variable considered separately for this purpose.

These driving factors, elsewhere suitably termed critical parameters, are chosen with assistance from subject matter experts, especially experts in the use and execution of the numerical solver under consideration. Thresholds can be placed upon these critical parameters to ensure that the resulting division of the parameter space reflects the desires of the experts for insight into particular aspects of the numerical solver output. For example, the value 0.3 above serves this threshold role for the parameter p_i. As shown in Fig. 2, these thresholds are used via the bisection method to generate the model input files. These files are then run through the numerical solvers to generate a potentially huge training sample database. This database, of course, is significantly smaller than what would have resulted from an exhaustive exploration of the full parameter space. However, even with the thresholds in place and bisection providing a more intelligent parsing of the input parameter space, the file can still be in the tens of millions to hundreds of millions of rows (one for each geometric point at each time-step for all possible variations in the critical parameters). How can one further perform data reduction on this database of training samples? Again, the priority is to reduce as much as possible and still retain representativeness of the underlying input parameter space.

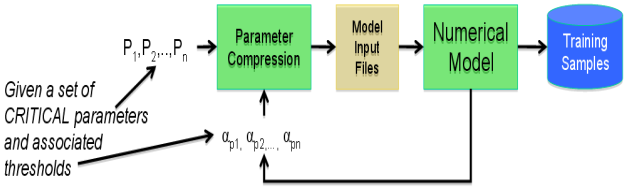

Fig. 2.

Another data reduction step can be tied to the geometry in which the solver is working. Through interaction with the subject matter expert, it may be decided that the complete boundary to the problem is essential to its solution, while of the interior points only a random 75 percent (or even 50 percent) is useful. Or perhaps only a portion of the boundary is truly essential, but more of the interior points are required to capture the details of the dynamics. These decisions should be made only in consultation with an experienced user of the numerical solver codes or with keen insights into the dynamics being modeled. However, the resulting reduction in the number of rows in the training database can be immense—potentially 30 to 40 percent. Similarly, there may be a few time-steps within the solver run that the user really wants to focus upon—the times immediately before and during an unusual event. These time-steps can be accentuated, while others, say, those occurring later that are less interesting, can be decimated through the use of a probability distribution on the time-steps. Those times that are desirable to keep can be associated with a 1 (i.e., 100 percent probability for inclusion in the reduced training set), while others, which

are less interesting, may be assigned any value between 0 and 1. The resulting distribution can then be run against the training samples database to determine which rows will be forwarded on to a reduced training samples database and which left behind. For instance, if over the full range of 100 time-steps, only the first 20 were absolutely essential for inclusion, and of the other 80, only 30 percent should be included, a distribution can be built so that time-steps 1 to 20 have a probability of 1 and time-steps 21 to 100 have a probability of 0.3. Then, on average, one should get (20 + 24) 44 time-steps in the reduced training sample, a reduction of 56 percent over the original database. Of course, it should be emphasized that caution should be exercised in all data reduction cases to maintain representativeness of the full range of parameters within that final reduced data set.

At this stage, the samples are divided into two subsets—one serving as a training set for the neural network, the other its test set. As mentioned above, the training set usually composes 90 percent of the parent set, with the remaining 10 percent going toward validation through the testing component. These percentages, although fairly canonical in the backpropagation literature, may vary slightly. As Fig. 3 shows, the neural network is trained until convergence criteria are met or its maximum number of epochs is completed. Then the test set is run through the network with its trained weights to determine how well the network can predict outputs on data it has never seen before. Here a key decision is made: Is the predicted output acceptable? If yes, the neural network can be treated as a surrogate model that can emulate the numerical solver within the ranges of its training. If not, however, several corrective paths lie open to pursuit. These options may include, from major to minor: (1) going back to the input of the numerical model and modifying the thresholds, since as they currently stand, they are ineffective at attaining the validated surrogate model,(2) revisiting and loosening the decisions made concerning inclusion and exclusion of geometry and time data points within the original training database, (3) altering the neural network algorithm from backpropagation to, say, recurrent networks, (4) changing the internal architecture of the current neural network by increasing the number of nodes within the hidden layer, and (5) modifying the neural network convergence criteria, forcing it into a tighter bound OR increasing the required number of epochs to execute during training. All of these possibilities require modification to some portion of the surrogate model construction process, some much more significant than others; however, some fix (perhaps a combination of some of the steps) is necessary since an invalidated surrogate model serves no practical purpose, and the user has gained no efficiencies by the additional steps. Some examples follow.

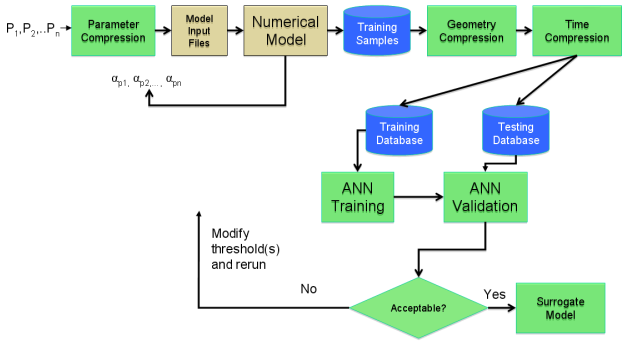

Fig. 3.

4. STUDY DOMAINS

Three trials of this technique have been conducted over widely divergent problem types. In the first, fairly simplistic proof-of-concept scenario, a two-dimensional model of the Herbert Hoover Dike was executed within surface-water/ground-water flow code (WASH123), where hydraulic conductivities and material types varied, to attempt predicting the flow at 21 cross-sections. In the second, a surface-water/ground-water code (ADaptive Hydraulics (ADH)) was used to quantify thermal effects caused by the effects of variations in albedo and the specific heat of solids within a three-dimensional model. The third involved using ADH to simulate the effects of a contaminant trace in a harbor, where the eddy viscosity and velocity varied.

4.1 Herbert Hoover Dike

This study case originated as a simple proof-of-concept exercise based on WASH3D runs of the Herbert Hoover Dike. The model had been executed under 11 material types and 3 hydraulic conductivity values, resulting in 33 input values. The outputs were just 21 points that quantified the cross-sectional flow. As this data had come from a design study, there were cases with and without a dike structure in place. While only 22 cases were available, which was far too few to build an adequate surrogate model, this small data set served remarkably well as a test-bed for performing sensitivity analyses over variations in backpropagation architectures (where the number of nodes in the hidden layer were varied) and the number of training epochs. The number of nodes in the hidden layer was varied from 15 to 70, and the number of training epochs varied from 100 to 10,000. Since the data sets were relatively small, this allowed for an exhaustive creation of neural networks to perform cross validation. In total, 4488 neural networks were built, and the results were compared globally using the standard deviation of the predicted to WASH3D output. One such model is illustrated in Fig. 4 showing approximately 2 percent error of predicted to actual.

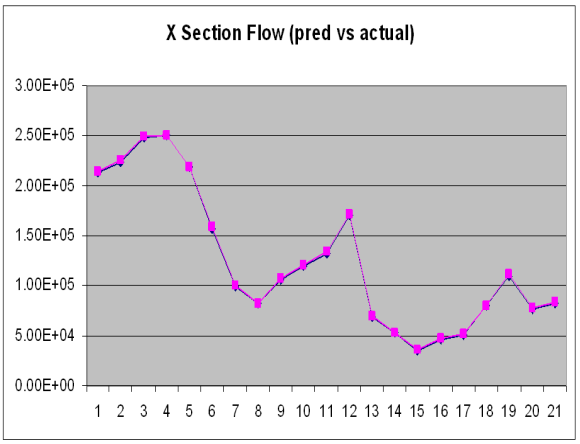

Fig. 4.

Again, this demonstration was created more as a sandbox to see if neural networks had anything to offer this scenario, and in so proceeding, resulted in the creation of numerous tools for performing sensitivity analyses of the output. Because of the relative low quantity of data, there was no need to perform bisection, or data reduction techniques, here.

4.2 3D Thermal Problem

Efforts to emulate the output of a three-dimensional thermal problem began the ongoing battle between an avalanche of data and its reduction to a manageable, yet representative amount. Rather than run this scenario exhaustively, the bisection code was utilized to batch-process the runs over variations in albedo and the specific heat of solids. Over the range of albedo and the specific heat of solids individually, the batch processing resulted in 81 distinct cases (which served as input to the neural network). In the preliminary stages, only one time-step was analyzed to see if a trained neural network could predict temperatures at a particular time, knowing only what similar albedo and specific heat of solids models were doing at that same time. As this severely reduced the size of the data set, no data reduction techniques were employed in these stages. Using the previously developed sensitivity analysis tools, neural networks were developed varying the number of training epochs and the number of nodes in the hidden layer (from 3 to 30 in multiples of 3), and turning momentum on and off. The results were evaluated at a set of salient points, chosen across the full range of points in space. This helped to determine the optimal neural network architecture (here, 6 inputs-12 nodes in the hidden layer-1 output) and the number of training epochs. Again, as this input set was rather small, 81 neural networks of this architecture were created (80 to train, 1 to test) to evaluate the error associated with the predictions. However, it was a static view, considering only a single time-step out of the whole range of time steps from the ADH run (Fig. 5).

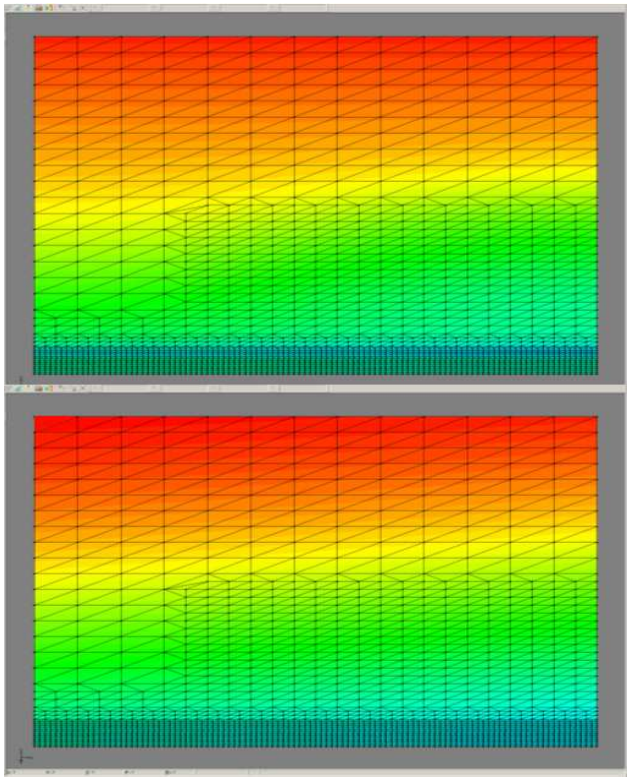

Fig. 5. This represents an oblique view of the 3D thermal problem, the upper showing ADH results and the bottom showing corresponding neural network output.

Further investigations broke out of this static view and considered multiple time-steps. Since the geometry of this scenario contained 13703 nodes, widening out the scope to multiple time-steps suddenly introduced millions of rows of (potential) input data to the neural network for training. Although bisection had been used in reducing the parameter space even in the static case described above, here further reductions proved necessary, such as sampling within the geometry space and focusing only upon the most interesting handful of the total time-steps available. This scenario then served as a sandbox to test out several innovative data reduction ideas. This case was still limited in that it attempted to predict the temperature at one time-step for a fixed albedo and fixed specific heat of solids pairing, given several previous time-steps for all 81 cases. Even at this level of surrogate model creation maturity, no attempt had been made to predict fully all time-steps of one case, given all other rows as a training set. This lack was later rectified by the Noyo contaminant trace scenario described below.

4.3 Contaminant Trace

A third domain also utilized the ADH code. It hypothesized that a pallet of a single contaminant had been released a few hundred meters above where the Noyo River flows out into the Fort Bragg Harbor to meet the Pacific Ocean. Tidal effects were turned off for these runs, as the study was to focus upon the river's effects as the water velocity and eddy viscosity values were varied. Although the geometry was much simpler than the thermal experiment above (having only 2689 nodes), the full suite of 48 time-steps quickly compelled usage of data reduction techniques. The bisection method (as described above) resulted in 90 ADH runs. The original database of training samples comprised 11.6 million rows. The geometry component was reduced by using all boundary nodes and 50 percent of the interior nodes. The time component was similarly reduced by retaining the first 33 time-steps, but using only 6 of the final 15. Taken together, these reduction efforts resulted in the training data set going from 11.6 million rows to 4.5 million rows, a reduction of approximately 62 percent. Of the 90 ADH runs, the reduced data from 89 are being used for training a backpropagation neural network, and the remaining one (located near the center of the parameter space) is being used to test. In other words, information from all 89 ADH runs over 39 time-steps are being used to train and predict the behavior for one unseen case for all 48 time-steps. As evidenced in Fig. 6, the dynamics for this case are rather complex—certainly not a strawman figure. Sensitivity analysis is currently underway for architectures with 30, 36, 44, and 88 nodes in the single hidden layer. Training, although time-consuming, will eventually provide a validated surrogate model for dynamics of the contaminant trace under widely-varying water velocity and eddy viscosity. Upon completion, this model can serve as an acceptable replacement for ADH within error bounds, and more importantly, can output answers within seconds in imitation of an ADH case that would take 2-3 hours to execute.

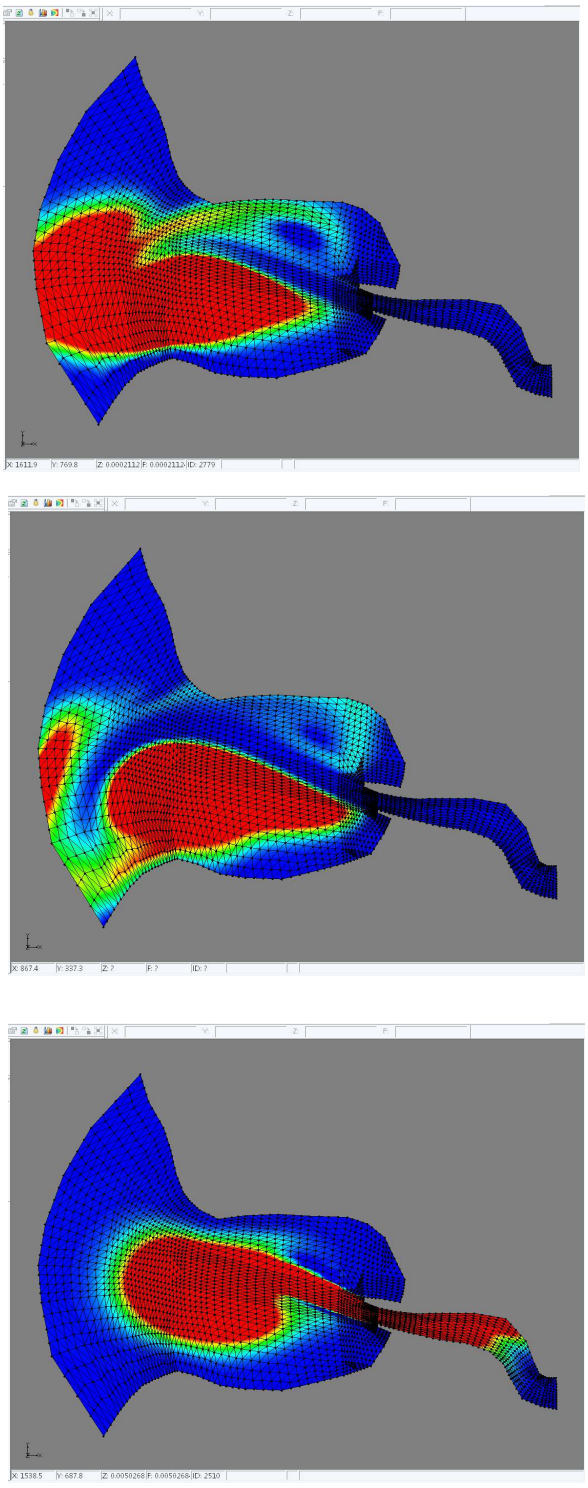

Fig. 6

5. BENEFITS ACCRUED BY OUR SOLUTION

Chief among the benefits of this technique is that actual HPC run time was reduced significantly, while still allowing the user the ability to predefine an acceptable level of error in output and adapt the bisection runs to acquire training set information appropriately. Sensitivity analysis at various levels of the investigation, in turn, reinforced confidence in the method.

Once a surrogate model is developed, it can be provided to a lay user to emulate the numerical models within the ranges of training. This does not require any specialized techniques or substantial experience in numerical modeling to use. Since a surrogate model can run and output its values within seconds as opposed to hours for a numerical model, it certainly demonstrates considerable time savings and lends itself much more readily to design optimization efforts and what-if analyses. A surrogate model typically runs on hardware platforms as unassuming as laptops and minimally configured personal computers. Specialized numerical

codes, on the other hand, often require resource-abundant environments (such as afforded by the high performance computing platforms), where costs can sometimes reach $30 per minute. Taken together, the ease of use, the speed of execution, and the relative low computing cost support the widespread use of these high-fidelity surrogate models in high-level planning efforts.

6. FUTURE WORK

Over the past 2 years, considerable effort has been expended to investigate the feasibility of using surrogate models as stand-ins for more complicated, more expensive, and harder-to-use numerical models. When difficulties arose, such as the data inundation that is a natural by-product of these types of codes, many paths were proposed and tested, and appropriate fixes were developed. At present, three challenges serve to focus the effort.

First, refining techniques to arrive at a "minimal" input set could still improve global performance. At present, the reductions are significant (on the order of 60-65 percent often); however, other non-bisection techniques (that may be model-specific) should be investigated as well. Although a minimal set may never be attained, the techniques should be implemented that allow for a more rapid asymptotic approach toward such a minimal set. Possibilities include statistical techniques and some data mining algorithms.

Second, many steps are still highly sequential in these methods. Some of this cannot be avoided—it is simply the nature of the problems. However, high on the list of priorities for future work is the need to refine the neural network training process to fully leverage parallel computing resources. Certainly the parallel nature of the HPC resources fit admirably well with the described sensitivity analyses.

Third, all portions of the surrogate model construction process will be transitioned to the HPC environment. As emphasized throughout, these types of problems usually involve very large data storage requirements. The data preparation step discussed above needs to be able to quickly derive the reduced input space i from I.

Success in any one of these areas could significantly improve the construction of surrogate models for faster utilization within high-level planning efforts. Together, the process could become much more automated, allowing the user to focus much more on different engineering designs under consideration and less on the computational tasks of data preprocessing.

CONCLUSIONS

As described, surrogate model construction offers a new, less expensive, more computationally efficient path in support of design optimization. Upfront, much effort has to be expended to generate a surrogate model that can emulate a numerical solver within given error bounds. Once done, however, SM_D can be used innumerable times to interpolate for unseen cases, which may be useful for design comparisons and what-if scenario analyses. This feature, in itself, relegates the exhaustive, brute force HPC runs to history. The learning curve for use of surrogate models is much flatter than for the typical numerical solver, which can require years of experience to master. However, for the pedantic, the correctness of a surrogate model can be demonstrated very quickly by comparison to any within-range numerical solver output previously unseen.

REFERENCES

Demuth, H., M. Beale, and M. Hagan, 2007: *Neural Network Toolbox 5 user's guide.* The MathWorks, Inc., Natick, MA. [Available online at http://www.agro.uba.ar/users/paruelo/redes/Matlab%20y%20ANNs/nnet_version%205.pdf.]

Hsieh, B. B., and T. C. Pratt, 2001: Field data recovery in tidal system using artificial neural networks (ANNs). Coastal and Hydraulics Engineering Technical Note CHETN-IV-38, U.S. Army Engineer Research and Development Center, Vicksburg, MS, 10 pp. http://chl.wes.army.mil/library/publications/chetn/

Mitchell, T., 1997: *Machine Learning*. Boston: WCB/McGraw-Hill, 414 pp.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, 1992: *Numerical Recipes in FORTRAN: The Art of Scientific Computing.* Cambridge University Press New York, 963 pp.

Rohwer, R., 1994: The Time Dimension of Neural Network Models. ACM SIGART Bulletin, **5**(3), *Special Section on Time in Neural Networks.* 36-44.

FUNDING

The research described in this paper was funded through the System-Wide Water Resources Program (SWWRP), Dr. Steven L. Ashby, Environmental Laboratory, U.S. Army Engineer Research and Development Center (ERDC), Program Manager. The authors would like to acknowledge the assistance of and express our thanks to Ms. Jackie Pettway, Coastal and Hydraulics Laboratory, ERDC, and Mr. David Stuart, Mr. Randall Hand, Mr. David Richards, Mr. Bobby Hunter, and Dr. Owen Eslinger, all of the Information Technology Laboratory, ERDC. Their assistance contributed greatly to the ongoing success of this project.